

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

Ben Verrill  
CS - 230 / 21EW4 Operating Platforms

Dr. Lyon  
March 19, 2021

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/15/21 | Ben Verrill | Initial Submission |
| 2.0 | 04/01/21 | Ben Verrill | Added OS Comparison Table |
| 3.0 | 04/12/21 | Ben Verrill | Added Recommendations |

**Instructions**

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_sbfa50wo7nsh)

Creative Technology Solutions will develop a web application for The Gaming Room based on their game Draw It or Lose It, which is only available on Android. The game consists of images being rendered to the screen as clues for teams to guess what phrase, title, or thing the images represents. The images will come from a large library of stock images. Each game lasts for four rounds, with each round taking up to one minute to complete. Drawings will take 30 seconds to fully render, during which time one team will have the opportunity to guess what it represents. If the team can’t guess the answer correctly, the other teams will each have a change to make one guess within 15 seconds.

The game will be developed for multiple platforms. It will support multiple players and teams. The names of the game and teams will be unique so as not to have the same names in use by multiple people at a time. There will only be one instance of a game being played at any given time. Each game, team, and player will have a unique id to enforce this.

We will enforce having only a single game in memory at a time by creating a GameService class that utilizes the Singleton pattern to keep track of if a game has been created and a list of individual instances of saved games that have been stored. We will use the Iterator pattern to iterate through our saved games list to check for game names and ids to ensure that only unique games are being retrieved or saved without exposing our data structures. We’ll have three classes to create games, players, and teams, and all will inherit common variables and methods from a parent class, Entity. Finally, we’ll have a class for testing to ensure we only have a single game in memory at a time and are properly utilizing our Singleton pattern.

## [Design Constraints](#_2et92p0)

* The game will be developed for as a web app for multiple platforms, but not for mobile. So we won’t be developing for iOS, and it’s already available on Android.

* Because it’s a web app, we need to decide where it will be hosted. We can choose from cloud hosting providers like AWS or Azure, or we can host it on prem.
* If we choose to host the game in a cloud environment, we need to determine how much storage and processing power we need to support multiple people/teams. We need to determine how much traffic will be utilizing the app to properly scope, and what sort of budget we have available to allocate to hosting services.
* If we choose to host in the cloud, we also need to decide what sort of database we’ll be using to store images and game data. This will affect what goes into our code. It also may affect budget, depending on the database and amount of storage used.
* If we choose to use our own infrastructure to host the app, we need to ensure we have the right hardware in place to manage the traffic coming into our data center. We may need to purchase additional servers and networking equipment to accommodate, which means we need to determine a budget to understand what we can allocate to equipment.
* Since we’re using a library of images for the game, we need to determine any fees involved with using the images and how that effects our budget.
* As a web app, we have multiple development stacks to choose from. This will determine the programming language we use. The stack chosen will also determine what sort of talent we’ll need to complete the project. This will affect the budget.
* Budget – The Gaming Room did not specify a budget for this project, but one will need to be determined in order to make decisions around staff, hosting, and other requirements.

## [System Architecture View](#_ilbxbyevv6b6)

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## [Domain Model](#_8h2ehzxfam4o)

We have an Entity class that has two instance variables that have been encapsulated within the class, a long named “id” and a String called “name”. This means that they have been set to private and cannot be directly interacted with outside of the using Entity’s class methods. This protects these variables from being modified outside of the Entity class. Next, we have two constructors. The first, Entity(), is set to private so that an Entity object cannot be created without its variables being set. The second is set to public and requires two parameters for the “id” and “name” variables. In Object Oriented Programming this is called overloading. We have the same name for the constructor, but this constructor will only be called if the right number and types of parameters are included. Finally, we have three public methods, getId(), getName(), and toString(). The getId() and getName() methods are our accessor methods, which in Object Oriented Programming allows us to retrieve data from our instance variables but not change them. The toString() method will allow us to display some data about our Entity class. These methods all return Strings when called.

Next, we have a Game class. This class inherits from the Entity class and, as a child class of Entity, has access to Entity’s variables and methods but also has attributes unique to it. This also means that Game objects are also Entity objects. The Game class has a single private variable “teams” which holds a list of teams created for the game object. This is encapsulated within the Game class and can only be accessed by the addTeam() method. Next, we have the public constructor for the Game class, which takes two parameters, a long variable called “id” and a String variable called “name”. We then have a public addTeam() method, which takes a String parameter called “name” and returns a Team object from the Team class. Finally, we have an overridden public toString() method that returns a String unique to the Game class. Because the Game class inherits the toString() method from the Entity class but overrides it to display different data, this is an example of the Object Oriented Programming principal of Polymorphism. The Game class has a zero to many association with the Team class, meaning that multiple instances of teams can be associated with a single game object and stored within its “teams” variable.

Next, we have a Team class. This class also inherits from the Entity class and takes on its attributes and methods. The Team class has a single private variable called “players” which stores a list of Player objects. This is encapsulated within the Team class and can only be accessed by the addPlayer() method. We then have the public constructor for the Team class which takes two parameters, a long variable called “id” and a String variable called “name”. Next, we have the public addPlayer() method which takes a String parameter called “name” and returns a Player object. Finally, there’s a public overriden toString() method which overrides our Entity class method to display data specific to the Team class. This makes the Team class Polymorphic. The Team class has a zero to many association with the Player class, meaning that multiple instances of players can be associated with a single team object and stored within its “players” variable.

Next up is the Player class. It, too, inherits from the Entity class. It has no unique attributes within it. The Player class has a public constructor that takes two parameters, a long variable called “id” and a String variable called “name”. Finally, there’s a public overridden toString() method which overrides our Entity class method to display data specific to the Player class. This makes the Player class Polymorphic.

We then have a GameService class. This class has five encapsulated variables within it. There is a private “games” variable that holds a list of Game objects. We then have three private long variables called “nextGameId”, “nextPlayerId”, and “nextTeamId”. Finally, we have a private “service” variable that holds a GameService instance. Because we only want to have a single game in memory at a time, we’ll be utilizing the Singleton pattern to ensure multiple games can’t be played at once. The Singleton pattern makes sure that there can be only one instance of a class, in this case the GameService class. This single instance will be what is stored in the “service” variable. To use the Singleton pattern, we make set the constructor for the GameService class to private. The constructor does not take any parameters. The constructor can only be accessed by the next method in the class, a public method called getInstance(). The getInstance() method will check to see if there is an instance of GameService stored in the “service” variable. If one isn’t, it will call the GameService constructor to create an instance and return that instance. Otherwise, it will return the instance of GameService already instantiated. This ensures that there’s only ever one game in memory at a time. We then have a public addGame() method that takes a String parameter called “name” and returns a Game object. Finally, we have five accessor methods. The public getGame() method takes a long as a parameter called “id” and returns a Game object. The next is also a public getGame() method, but its method is a String called “name”. This is an example of overloading a method, and the method called is determined by the type and number of parameters contained within the call. This method also returns a Game object. The public getGameCount() method returns an integer and takes no parameters. The public getNextPlayerId() method returns a long and takes no parameters. Finally, the public getNextTeamId() method returns a long and takes no parameters. The GameService class has a zero to many association with the Game class, meaning that multiple instances of games can be associated with the single GameService object and stored within the games list.

Finally, we have a ProgramDriver class and a SingletonTester class. The ProgramDriver class contains the public main() method that runs our application. It has a directed association with the SingletonTester class, where it uses the SingletonTester’s public method testSingleton() to make sure that only one instance of GameService is running at a time.



## [Evaluation](#_2o15spng8stw)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | MacOS Server is an OS created by Apple that’s made easily available for download on the App Store. It’s three primary use cases center around directory services, mobile device management, and shared network storage. It’s made to be able to easily integrate with other operating systems by Apple, like iOS. However, it doesn’t seem to be built with web application hosting in mind. While researching web application hosting on Apple’s Support Community, most people seem to find it easier to host web services on client OS versions rather than macOS Server, and you’d still need to set up a Linux server to run on the client, which seems to defeat the purpose. While not expensive at $20, the server is not free though it does come with support. | One of the best parts about using Linux is that it’s free to use. Linux is open source and widely used, which means the Linux community is constantly adding features and solutions that can help more efficiently build and run web applications. It also tends to have stronger security built in because it’s requires more effort for executables to be triggered, which makes virus and malware less effective. Because it’s open source, that also means that administrators have access to all processes and functions of the OS, which gives them the greater flexibility to tweak the source code as needed. This can also be considered one of the cons, however, because it can make setting up a Linux server more complicated and require more technical administration. While using the OS is free, generally support is not depending on the distribution being used. | Windows is an operating system owned by Microsoft. While it’s relatively expensive to use, it comes with support and regular updates and, because many businesses are already using Windows Server for things like Active Directory, business may already have access to the web server portion, Microsoft IIS. Windows users typically have access to an intuitive to use GUI and don’t need to rely on Command Line to work with the server, making administration potentially easier. Windows tends to be better support by third party software. There are more security risks, however, and the fact that Windows is not open source gives less flexibility for admins to tweak the software. | Hosting web application servers on mobile operating systems can be done, but it doesn’t seem very efficient. For iOS, one could download iSH Linux Shell and the build a web server using Python. However, this would only work for simple websites, not the type of game we’re looking to build that requires many people interacting with an application. With Android one could use HTTP Server from Apache. This would provide functionality closer to what we’re looking for, but we would be limited to the resources provided by the mobile device (cpu, storage, etc.) and this would not be an efficient way to run a web application. While running web apps on a mobile OS is possible, there don’t seem to be any real benefits to doing so. |
| **Client Side** | The client for our web application will be a web browser. The most used browsers as of January 2021 are Chrome, Safari, Edge, Firefox, Opera, and Internet Explorer. Of these, MacOS supports all except Internet Explorer. The web application needs to be compatible with each web browser we want to make it available on, and the more we add means more time and money that our front-end developer will need to do so. We could lower costs by choosing a framework that already supports multiple browsers, such as React, Angular, or Vue.js. Here, again, we need to decide what browsers we want as clients because some frameworks don’t work with all of them (Angular doesn’t work with Opera). Introducing new frameworks also means that we now need front-end developers with expertise that may differ from what we’re using on the backend, as the ones mentioned are all JavaScript based. Testing will still need to be done for each browser to ensure a consistent experience across browsers, but using frameworks with multiple browser support will cut down on development time. Customizing the experience for each browser can also be done by using a Doctype for HTML files and customizing CSS rules for each browser. | The client for our web application will be a web browser. The most used browsers as of January 2021 are Chrome, Safari, Edge, Firefox, Opera, and Internet Explorer. Of these, MacOS supports all except Internet Explorer and Safari. The web application needs to be compatible with each web browser we want to make it available on, and the more we add means more time and money that our front-end developer will need to do so. We could lower costs by choosing a framework that already supports multiple browsers, such as React, Angular, or Vue.js. Here, again, we need to decide what browsers we want as clients because some frameworks don’t work with all of them (Angular doesn’t work with Opera). Introducing new frameworks also means that we now need front-end developers with expertise that may differ from what we’re using on the backend, as the ones mentioned are all JavaScript based. Testing will still need to be done for each browser to ensure a consistent experience across browsers, but using frameworks with multiple browser support will cut down on development time. Customizing the experience for each browser can also be done by using a Doctype for HTML files and customizing CSS rules for each browser. | The client for our web application will be a web browser. The most used browsers as of January 2021 are Chrome, Safari, Edge, Firefox, Opera, and Internet Explorer. Of these, Windows supports all of them. It should be noted, though, that Apple no longer provides support for Safari on Windows, so the most up to date versions can’t be used. The web application needs to be compatible with each web browser we want to make it available on, and the more we add means more time and money that our front-end developer will need to do so. We could lower costs by choosing a framework that already supports multiple browsers, such as React, Angular, or Vue.js. Here, again, we need to decide what browsers we want as clients because some frameworks don’t work with all of them (Angular doesn’t work with Opera). Introducing new frameworks also means that we now need front-end developers with expertise that may differ from what we’re using on the backend, as the ones mentioned are all JavaScript based. Testing will still need to be done for each browser to ensure a consistent experience across browsers, but using frameworks with multiple browser support will cut down on development time. Customizing the experience for each browser can also be done by using a Doctype for HTML files and customizing CSS rules for each browser. | The client for our web application will be a web browser. The most used browsers as of January 2021 are Chrome, Safari, Edge, Firefox, Opera, and Internet Explorer. The majority of users for mobile operating systems fall into either Android or iOS, so these will be the two operating systems we’ll consider. All of the above browsers are supported for both iOS and Android, with the exception of Internet Explorer on iOS. The web application needs to be compatible with each web browser we want to make it available on, and the more we add means more time and money that our front-end developer will need to do so. We could lower costs by choosing a framework that already supports multiple browsers, such as React, Angular, or Vue.js. Here, again, we need to decide what browsers we want as clients because some frameworks don’t work with all of them (Angular doesn’t work with Opera). Introducing new frameworks also means that we now need front-end developers with expertise that may differ from what we’re using on the backend, as the ones mentioned are all JavaScript based. Testing will still need to be done for each browser to ensure a consistent experience across browsers, but using frameworks with multiple browser support will cut down on development time. Customizing the experience for each browser can also be done by using a Doctype for HTML files and customizing CSS rules for each browser. |
| **Development Tools** | There are many programming languages that can be used for web application development. The more popular are JavaScript, Python, Java, C++, and PHP. Each have frameworks and libraries that can be used to more efficiently build web apps. For example, JavaScript has React, Java has Spring, and Python has Django. All are operating system agnostic. Because the client for our application is the web browser, we only need to consider which web browsers can be used for MacOS when thinking about deployment options. Some of the most popular IDEs for developing web apps are Visual Studio Code, Atom, and IntelliJ IDEA. All are compatible with MacOS, Windows, and Linux and support the above programming languages. Visual Studio Code has a plethora of extensions that can be downloaded for more efficient coding and error handling and is a free IDE. Atom is run by Github and has the support of the Github community and is very customizable. It is also free. IntelliJ is primarily used for Java, but supports other languages as well, and is able to automatically add tools according to context to make coding more efficient. It is free, but also has a paid version with more tools for enterprise development. All of these IDEs also have git integration, making deployment easier. | There are many programming languages that can be used for web application development. The more popular are JavaScript, Python, Java, C++, and PHP. Each have frameworks and libraries that can be used to more efficiently build web apps. For example, JavaScript has React, Java has Spring, and Python has Django. All are operating system agnostic. Because the client for our application is the web browser, we only need to consider which web browsers can be used for Linux when thinking about deployment options. Some of the most popular IDEs for developing web apps are Visual Studio Code, Atom, and IntelliJ IDEA. All are compatible with MacOS, Windows, and Linux and support the above programming languages. Visual Studio Code has a plethora of extensions that can be downloaded for more efficient coding and error handling and is a free IDE. Atom is run by Github and has the support of the Github community and is very customizable. It is also free. IntelliJ is primarily used for Java, but supports other languages as well, and is able to automatically add tools according to context to make coding more efficient. It is free, but also has a paid version with more tools for enterprise development. All of these IDEs also have git integration, making deployment easier. | There are many programming languages that can be used for web application development. The more popular are JavaScript, Python, Java, C++, and PHP. Each have frameworks and libraries that can be used to more efficiently build web apps. For example, JavaScript has React, Java has Spring, and Python has Django. All are operating system agnostic. Because the client for our application is the web browser, we only need to consider which web browsers can be used for Windows when thinking about deployment options. Some of the most popular IDEs for developing web apps are Visual Studio Code, Atom, and IntelliJ IDEA. All are compatible with MacOS, Windows, and Linux and support the above programming languages. Visual Studio Code has a plethora of extensions that can be downloaded for more efficient coding and error handling and is a free IDE. Atom is run by Github and has the support of the Github community and is very customizable. It is also free. IntelliJ is primarily used for Java, but supports other languages as well, and is able to automatically add tools according to context to make coding more efficient. It is free, but also has a paid version with more tools for enterprise development. All of these IDEs also have git integration, making deployment easier. | There are many programming languages that can be used for web application development. The more popular are JavaScript, Python, Java, C++, and PHP. Each have frameworks and libraries that can be used to more efficiently build web apps. For example, JavaScript has React, Java has Spring, and Python has Django. All are operating system agnostic. Because the client for our application is the web browser, we only need to consider which web browsers can be used for iOS and Android when thinking about deployment options. Some popular IDEs for Android are Android AIDE and Android Web Developer. AIDE supports Java, C++, JavaScript, HTML5, and CSS. I didn’t see any support for automation tools or other frameworks, but does have a debugger and supports error checking. It looks to be free. Android Web Developer supports HTML, PHP, JavaScript, and HTML/CSS. It allows for remote access through FTP, FTPS, SFTP WebDAV. It supports code efficiency tools like code completion detection, code beautifying, and error checking. It has both a free and paid version. Both support git integration. I didn’t find any IDEs for iOS for web app development. All the ones in the iOS store were for learning purposes. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: We recommend hosting the web application servers in a cloud environment, specifically in AWS. We would use Amazon Elastic Cloud Computing for hosting the game. AWS EC2 will give us the ability to quickly spin up new web servers as our game becomes more popular. It allows for flexibility for resource management so that we can adjust needed memory and storage on the fly and allows for keeping maintenance costs lower by only charging for capacity that’s used. Using a hosted environment also lowers maintenance costs by shifting responsibility of availability (SLAs are committed to 99.99% uptime) and updates to the cloud provider. This means we won’t need to maintain any physical data centers or need any centralized physical location for employees to work. Our work environment can be accessed anywhere there’s internet availability. Using a cloud provider also allows for additional tools that can be used that are offered by the provider as the game needs to scale. These include security options and analytics (Amazon, 2021).  
     
   For the tech stack, we recommend using MEAN. This means MongoDB for the database, Express JS for the web framework, Angular JS for the front-end, and Node JS for the runtime environment. One benefit using MEAN is that both front-end and back-end code is in JavaScript, limiting the amount of specialization we need for developers (Harbor & Holmes, 2019).

1. **Operating Systems Architectures**: We’ll recommend Linux as the operating system of choice for the web server. It will give us the control to tweak services within the OS to adjust for performance and is easily scalable as our user base grows. Specifically, we’ll use Ubuntu as our distro. It’s free to use compared to Windows. As the user base grows and server needs become more complicated, we could then choose to purchase support for Ubuntu at $500 per year for virtual machines, which is still nearly half the cost of a standard Windows Server license (Canonical, 2021).
2. **Storage Management**: For the Ubuntu OS, we need a minimum of 2.5GB of storage on the host server (Canonical, 2021). Our game code should be relatively small, with most of our storage requirements needed for image files. With 200 image files at 8MB per file, we need an additional 1.6GB of storage space. I could not find the system requirements for NodeJS, but the installer file was around 30MB (OpenJS Foundation, 2021). Our initial storage needs are just over 4GB, so we recommend 6GB at a minimum to start. We’ll need the ability to scale our storage needs as our user base grows, as well as if we decide to grow our image database. AWS allows us to do this automatically. As for where we’ll store the actual data, we’ll store the structured data like saved games and user and team information in the database. Our code files and image files will be stored directly on the web server. As our user base grows, we can spin off the database to its own separate server as needed, but for performance reasons it makes the most sense to keep the image files on the web server considering their storage size.
3. **Memory Management**: For the Ubuntu OS, we need a minimum of 1GB of RAM on the host server. The MongoDB database requires a minimum of 256MB of RAM (MongoDB, 2021). We don’t know how many games will be running concurrently at the time of release, but if we assume every image is being accessed at the same time, we’ll need 1.6GB of RAM at a minimum to temporarily store them. This gives us a minimum of approximately 2.9GB of RAM needed. We recommend staying below a threshold of 80% max RAM utilization, so the server should have at least 4GB of RAM initially. We’ll need the ability to scale our memory needs as our user base grows. AWS allows us to do this automatically.
4. **Distributed Systems and Networks**: We recommend that the game be delivered to users through the following web browsers: Chrome, Safari, and Firefox. This will allow for 85% coverage for internet users (“Desktop Browser Market Share Worldwide, 2021). This will mean we don’t have to worry about developing for specific operating systems players use, because the majority will be able to use one of the three chosen browsers. The game will not be playable if the player’s local internet connection is down. However, we can mitigate downtime within the AWS environment by using multiple instances of our web server in different regions of the world to achieve greater high availability. AWS has a SLA of 99.99% uptime, and includes load balancing in its services. It automatically replicates data within Availability Zones to prevent data loss from component failure (Amazon EC2, 2021). By reducing the risk of a single point of failure, we decrease the risk of downtime where players can’t use the game. The other benefit of having multiple instances of the web server in different geographical regions is that content can be delivered to players within those regions faster. If we have a server in Virginia and a server in Bangladesh, a user in India won’t experience additional latency issues by having to connect with the US server as they can use a more local server. Finally, because the application is hosted in AWS, AWS handles the maintenance of the servers and networks. We won’t need to care for physical data centers or incur additional costs in hardware and labor by hosting on our own.
5. **Security**: There’s no mention within the requirements of there being any costs involved for players to play the game. This means that there’s not much in the way of sensitive user information to protect. At the very least, we’ll require the user to provide a username and password to log into the game. The password will be at least 8 characters long and contain uppercase characters, lowercase characters, digits, and special characters to achieve greater password security (Weiss, 2020). These will be hashed and the hash will be stored in the database so no actual password is accessible to outside threats. We’ll provide the option for a checkbox for the browser to remember the user, which places a security token on the user’s browser that can be checked each time the user tries to access the game. This will allow the user to not have to sign in every time. When a user signs in, we’ll use role-based access control to determine what they’re able to do. This will limit risk to proprietary company information if a user’s profile is breached. We’ll also implement security features like input validation for any text fields the player can utilize. We'll limit the number of characters and character type depending on the use case of the field. This prevents from various overflow and injection style attacks. For administrators accessing the application, we’ll provide stronger security controls. For accessing the app, they’ll need a username and password, but we’ll also require multi factor authentication for logging in. This is because we’ll assign admins a role that gives them greater access to the application and thus to more sensitive company information. We’ll take a multi-tiered approach, where admin’s will only have access to what they need to get their jobs done. Perhaps this means some can upload files to the web server such as new images or updates to code, but others cannot because their job is strictly to manage the server instances in AWS. This implements the idea of least privilege, providing greater security. We’ll utilize antivirus solutions to analyze files uploaded to the server to ensure additional protection from outside threats. Finally, we can place a firewall in front of the web server to minimize the type of traffic allowed to get through to the server.

**References**

Amazon EC2. (2021). Retrieved 2021, from <https://aws.amazon.com/ec2/>

Apple Support (2021). MacOS server user guide. Retrieved April 02, 2021, from <https://support.apple.com/guide/server/welcome/5.11/mac>

Canonical (2021) Retrieved 2021 from <https://ubuntu.com/advantage>

Desktop browser market share worldwide. Retrieved April 04, 2021, from <https://gs.statcounter.com/browser-market-share/desktop/worldwide/#monthly-202101-202101-bar>

Harber, C. & Holmes, S (2019). *Getting MEAN with Mongo, Express, Angular, and Node, Second Edition*. Shelter Island, NY. Manning Publications Co.

OpenJS Foundation (2021) Retrieved 2021 from <https://nodejs.org/en/download/>

Mobile Operating System market share worldwide. Retrieved April 04, 2021, from <https://gs.statcounter.com/os-market-share/mobile/worldwide>

MongoDB (2021) Retrieved 2021 from <https://docs.mongodb.com/manual/administration/production-notes/>

Thomas, O. (2020). *Windows Server 2019: Inside out*. Redmond, WA: Microsoft Press, a division of Microsoft Corporation.

Turnbull, J., Matotek, D., & Lieverdink, P. (2017). *Pro Linux System Administration: Learn to Build Systems for Your Business Using Free and Open Source Software, Second Edition*. Apress.

Weiss, M. (2021). *CompTIA Security+ SY0-601 Exam Cram*. Pearson.